Още задачи върху графи

Тодор Дуков

Най-къси пътища в тегловен граф

Започваме с може би най-известния нетривиален графов алгоритъм – алгоритъмът на Дийкстра за намиране на най-къси пътища в тегловен граф. При него започваме с един стартов връх, и намираме най-късите пътища между този стартов връх и всеки достижим от него. Ето как става това:

```
vector<int> dijkstra(vector<vector<pair<int, int>>> g, int start)
{
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    vector<unsigned long long> dists(g.size(), INF);
    vector<int> parents(g.size(), -1);
    dists[start] = 0;
    pq.push({0, start});

    while (!pq.empty())
    {
        int curr = pq.top().second;
        int dist = pq.top().first;
        pq.pop();

        for (const pair<int, int> &edge : g[curr])
        {
            int adj = edge.first;
            int weight = edge.second;

            if (dist + weight < dists[adj])
            {
                parents[adj] = curr;
                dists[adj] = dist + weight;
                pq.push({dist + weight, adj});
            }
        }
    }

    return parents;
}
```

Алгоритъмът има сложност по време $\Theta((|V| + |E|)\log(|V|))$ в най-лошия случай.

Структурата Union-find/Disjoint-union

Ще разгледаме метод за поддържане на разбиване на множества от вида $\{0, \ldots, n\}$. Искаме да започнем от разбиването $\{\{0\}, \ldots, \{n\}\}$, и след това бързо да можем да обединяваме множества и да проверяваме дали някои i, j попадат на едно и също място в разбиването. За да може тези проверки и сливания да стават бързо, се използва структурата Union-find (понякога се нарича Disjoint-union). Ще имаме единствена функция unify(i, j), която приема $i, j \in \{0, \ldots, n\}$ и слива множествата от разбиването, в които i и j участват. Ако преди това те са били в едно и също множество, то тогава накрая връщаме `false`, иначе връщаме `true`.

Имплементацията е следната:

```
struct union_find
{
private:
    vector<int> leaders;
    vector<int> sizes;

    int get_leader(int x)
    {
        while (x != leaders[x])
        {
            leaders[x] = leaders[leaders[x]];
            x = leaders[x];
        }

        return x;
    }

public:
    union_find(int n) : leaders(n), sizes(n)
    {
        for (int i = 0; i < n; ++i)
        {
            leaders[i] = i;
            sizes[i] = 1;
        }
    }

    bool unify(int x, int y)
    {
        x = get_leader(x);
        y = get_leader(y);

        if (x == y)
            return false;

        if (sizes[x] < sizes[y])
            swap(x, y);

        leaders[y] = x;
        sizes[x] += sizes[y];

        return true;
    }
};
```

Сложността на извикването на unify е $O(\alpha(n))$, където α е обратната функция на Акерман. Тази функция расте изключително бавно – $\alpha(n) \leq 4$ за $n < 10^{600}$.

Алгоритъмът на Крускал за намиране на минимално покриващо дърво

Графът ще пазим като масив от ребра:

```
struct edge
{
    int from;
    int to;
    int weight;
    friend bool operator<(const edge &e1, const edge &e2) { return e1.weight < e2.weight; }
};
```

Алгоритъмът на Крускал работи по доста естествен начин. Гледаме да приоритизираме ребрата с най-малки тегла. Не да ги добавяме само ако биха образували цикъл.
Имплементацията е следната:

```
vector<edge> kruskal(vector<edge> edges, int n) // приемаме, че върховете са от 0 до n
{
    sort(edges.begin(), edges.end());
    vector<edge> mst;

    union_find dsu(n);

    for (const edge &e : edges)
    {
        if (dsu.unify(e.from, e.to))
            mst.push_back(e);
    }

    return mst;
}
```

Сложността на алгоритъма по време е $\Theta(|E|\log(|E|))$ в най-лошия случай.

Задачи

Задача 1. Да се напише алгоритъм, който при подаден ориентиран тегловен граф намира теглата на минималните пътища между всеки два върха.

Задача 2. Да се напише алгоритъм, който при подаден тегловен ориентиран ацикличен граф и негов начален връх намира най-късите пътища от този начален връх до всички достижими от него.

Задача 3. Да се напише алгоритъм, който при подаден свързан цикличен граф с ребро, чието премахване ще превърне графа в дърво, връща това ребро.

Задача 4. Нека е дадено крайно множество от променливи $X = \{x_0, \dots x_k\}$. Уравнения над X ще наричаме всички низове от вида $x_i = x_j$ и $x_i \neq x_j$ за $0 \leq i, j \leq k$. Да се напише алгоритъм, който при подадено множество от променливи X и списък $E[1 \dots n]$ от уравнения над X проверява дали системата, съставена от списъка с уравнения има решение.

Задача 5. Алис и Боб имат граф с върхове измежду 0 и n и три типа ребра:

- ребрата от тип 1 могат да бъдат траверсирани само от Алис
- ребрата от тип 2 могат да бъдат траверсирани само от Боб
- ребрата от тип 3 могат да бъдат траверсирани и от двамата

Да се напише алгоритъм, който при подаден масив от ребра $E[1 \dots k]$ т.е. тройки от типа $\langle type, i, j \rangle$ за някои $type \in \{1, 2, 3\}$ и $i, j \in \{0, \dots, n\}$ връща максималния брой ребра, които могат да се махнат, и графът пак да може да бъде обходен от Алис и Боб. Ако някой от двамата не може да обходи първоначалния граф, да се върне -1.

Задача 6. За всеки две точки $p_1 = (x_1, y_1)$ и $p_2 = (x_2, y_2)$ в $\mathbb{Z} \times \mathbb{Z}$, цената на свързване на p_1 и p_2 ще бъде $|x_1 - x_2| + |y_1 - y_2|$. Да се напише алгоритъм, който при подаден масив $P[1 \dots n]$ от точки намира минималната обща цена на свързване, за която между всеки две точки от $P[1 \dots n]$ ще има път.